Seminar: Technology of Autonomous Vehicles

Topic 1: Lane Detection

Fazli Faruk Okumus(737548), Mevluede Tigre(737551)

Contents

Abstract

This paper gives technical background about our 2 different implementations (Simple and Advance Methods) of lane detection and their comparable results. In the simple method, Canny Edge Detection and Hough Transformation are used, whereas Perspective Transform, and Sliding Window are used for the advanced version. However, both of them highly depend on the visibility of lane lines and certainly depend on the environmental conditions. Implementations are tested with three different videos that were taken in different conditions. The simple method works well for non-curved and non-changed conditions, but its accuracy is really low. To improve the accuracy in harsh conditions, an advanced algorithm has been implemented, which has better performance, but high process time.

1 Introduction

In recent years, autonomous vehicles have gained speed and importance with today's technology. Lane detection and lane-keeping are among the most important building blocks of autonomous driving to enable perception, which is also used in ADAS (Advanced Driver Assistance System). Detecting the lanes provides driving support and localization information to vehicle control using camera

images. A person can involuntarily (reflexively) hold the car between the lanes while driving. But getting this done on a computer is actually quite difficult and complex. Why is it so? The first reason is that the steps to be implemented are quite a lot. First of all, various computer vision techniques should be applied to the image taken from the cameras placed in the vehicles, which are including converting RGB to Grayscale(reducing the amount of data without losing the important information), perspective transformation(getting vision similar to the bird-eye view on the road) and more. The second reason is to use complex algorithms. In this project, we applied some complex computer vision algorithms like Canny Edge Detection, Hough Transformation, Perspective Transform, and Sliding Window by means of the common open-source library called OpenCV and Python.

2 Background

2.0.1 Convert the image RGB to Gray Scale

RGB image has to be converted to grayscale format to be used in Canny Edge Detection that accepts only grayscale images. RGB image has 3 different channels each of them represents the main color Red-Green-Blue. Grayscale image, on the other hand just consists of 8 Bit channel that 0 represents black and 255 represents white. Any noise elimination methods are not implemented on the input frame, because Canny Edge Detection applies a Gaussian kernel to get rid of the noise in the image, otherwise informants and key points can be lost in the image.

Figure 1: Conversion RGB to Gray Scale image

2.0.2 Define Region of Interest (ROI)

A region of interest is a place on the image where interesting objects can be searched. To be able to identify lanes on the image, a region of interest on the image should be established, cause it helps to narrow the interesting area. In the case study, lanes are not looked for in the sky or in all the way to the right or the left. ROI coordinates were manually set.

Figure 2: ROI

2.0.3 Convert the image RGB to HLS Color Space

Colors can be represented by different methods.
One of them is RGB(Red-Green-Blue) color space. But one color can be also represented by its Lightning, Saturation, etc. This can be very useful for different conditions and for extracting information and some parts from the image.

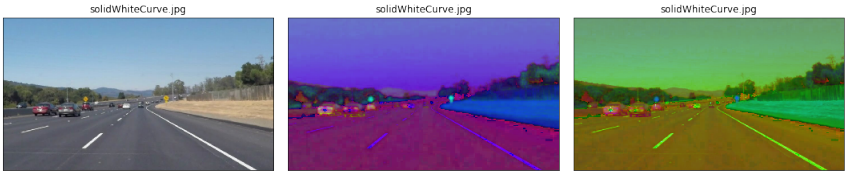


Figure 3: Different representations of images with different Color spaces

The main idea of the advanced algorithm is to extract lane lines as well as possible. Because these pixels are used to detect lanes and calculate curvature. To extract lane lines, the characteristics of the lines have to be understood. Lines are white or yellow and most of the time they are evident and smooth. Some tests are executed on the test images to find out which color space and which channel of it is more useful for extracting lane lines.

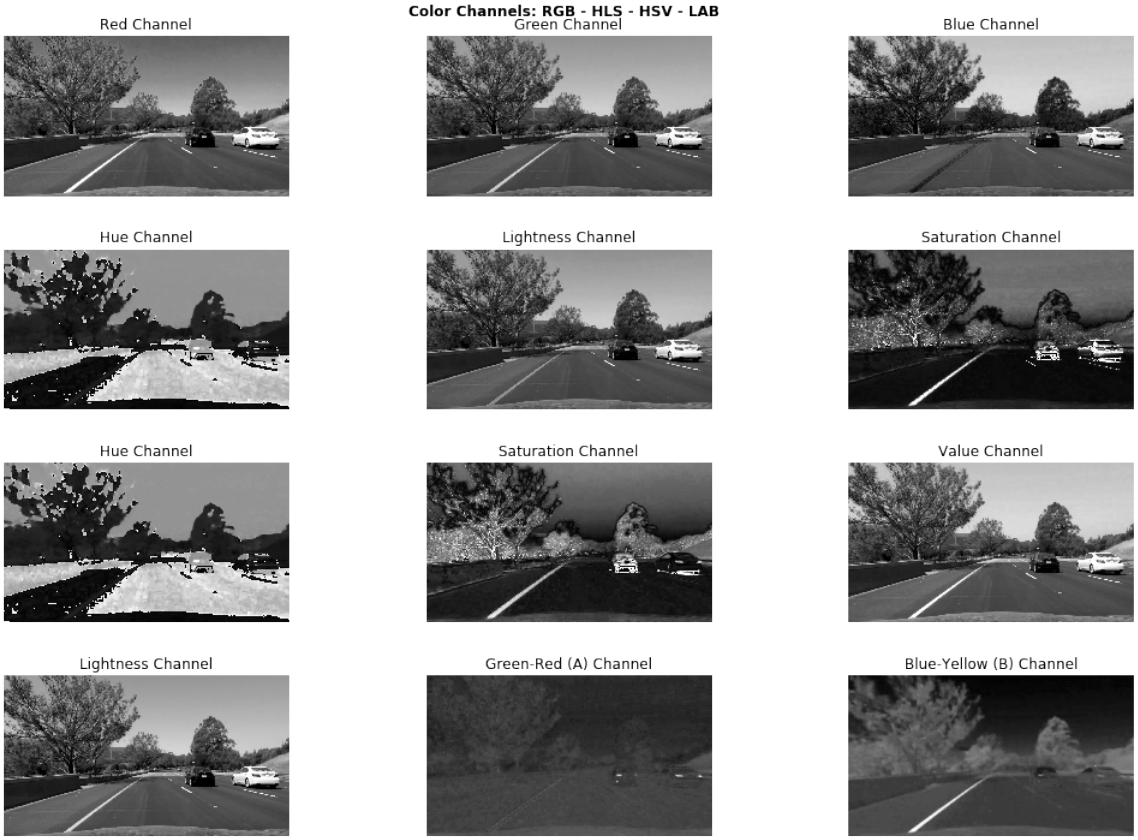


Figure 4: Representation of lane lines in different color spaces [1]

As you can see in Figure 4, saturation channels of HSV and HLS color spaces give very good results, despite the noisy output of the Hue channel. Also, the Lightness channel is giving grayish image and is good for identifying both yellow and white lane lines. Additionally, some thresholding values have been found from [2] for HLS color space to extract white and yellow colors. Because of these reasons, this color space has been used.

2.0.4 Perspective Transform

ROI area in the 2D image is converted to a birds-eye view through a perspective transform technique (**cv2.getPerspectiveTransform()**, **cv2.warpPerspective()**) like in the image as shown in Figure 5.

The input image is a two-dimensional matrix and each pixel has a value of 0-255 for an 8-bit grayscale image. In the transformation, the location of each pixel will transform to a new position by using a transformation matrix which can be found by using specified source(ROI) and destination points that are set by hand.

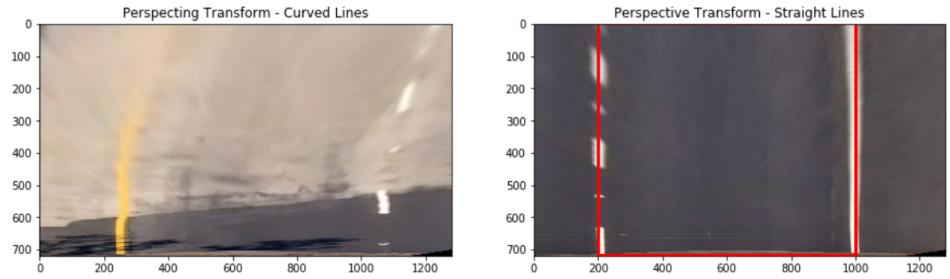


Figure 5: Perspective Transform Illustration [2]

3 Methods and Algorithms

3.1 Simple Lane Detection Algorithm

In this section, the first algorithm is introduced which is based on common computer vision methods to find lane lines. This is actually a simple and basic algorithm because based on Hough transformation which is specialized to find linear lines and so, will not perform so well on curves. The general structure of the algorithm and processes that apply to frames can be seen in Figure 6.

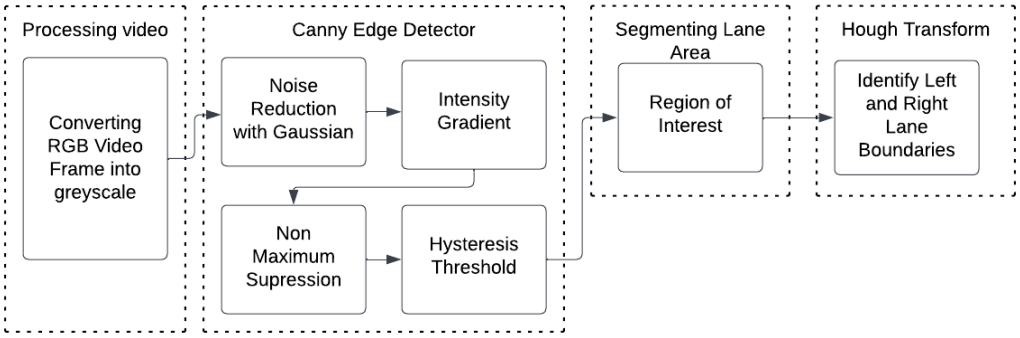


Figure 6: Simple Algorithm Structure

3.1.1 Canny Edge Detection

Applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed by preserving the important structural properties of an image. For implementation, the Canny Edge Detection algorithm was chosen. Canny Edge Detection, which was developed by John F., is a popular edge detection algorithm and consists of 4 stages Noise Reduction, Intensity Gradient Calculation of the image, Non-maximum Suppression, and Hysteresis Thresholding. This algorithm works with grayscale pictures and that is why the image is needed to be converted to grayscale before following the stages.

- **Noise Reduction:** Since edge detection is precision to noise in the image. To eliminate the noise, a Gaussian filter can be applied and it gives blurs and smoother images. Basically, the smallest the kernel, the less visible the blur [1]. In this project, a 5x5 Gaussian Kernel was applied, causing OpenCV **cv.Canny()** function use it as a default.
- **Intensity Gradient Calculation:** In this step, after applying Sobel filters that highlight intensity change in both directions(edge intensities: horizontal I_x and vertical I_y) on the smoothened image, the edge gradient and direction for each pixel are detected respectively as follows:

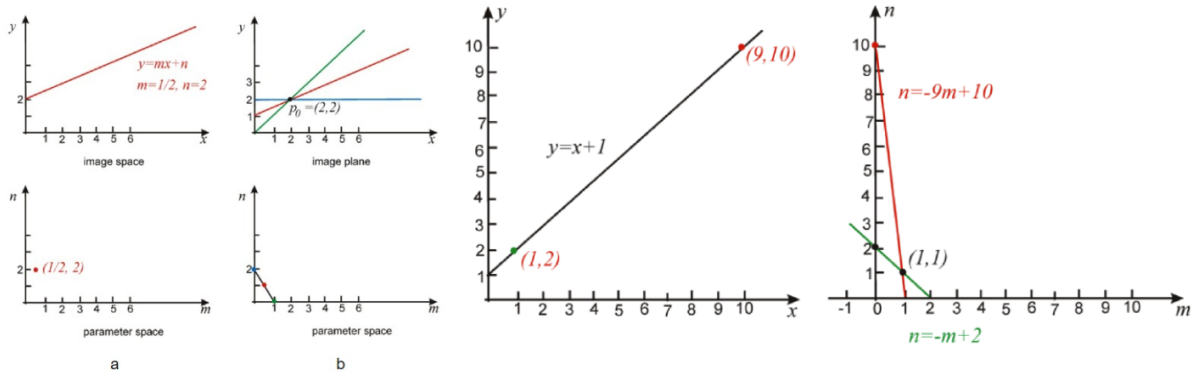
$$\|\nabla I\|_2 = \sqrt{I_x^2 + I_y^2} \tag{1a}$$

$$\Theta = \arctan \frac{I_y}{I_x}, \tag{1b}$$

- **Non-maximum Suppression:** In the blurred image, some of the edges are thick and others are thin. To eliminate the thick edges, any undesirable pixels that can not compose the edge are disregarded.
- **Hysteresis Thresholding:** This stage decides whether the edges are certain edge or not. There are two types of thresholds as low and high. If an edge has an intensity gradient higher than a high threshold, it is accepted as a certain edge. Whereas the intensity gradient of an edge is lower than the low threshold, it is accepted as non-edge and the edges in this category are disregarded. By doing this method, strong edges can be found.

3.1.2 Hough Transformation

Hough transformation can be applied to the pixels found by Canny Edge Detection to find lanes' lines. The theory behind the algorithm is that line in the image space can be expressed with two variables that are slope m and intersection n which corresponds to a point in parameter space. A point in the image space, on the other hand, can be represented with a line in parameter space because infinitely many lines can pass through this point, and variation in slope and intersection parameters corresponds line in the parameter space. The equation $y = mx + n$ is satisfied for each x, y point belonging to this line. The line in the image space is defined via the intersection of lines in the parameter space that are associated with co-linear points found by edge detection on line image space.



(1) On the left, the line in the image space corresponds to the point in the parameter space.

(2) Intersection of Lines corresponds a Line in Image Space

Figure 7: 1: Duality of the image and parameter space is shown 2: Intersection of the lines in the parameter space corresponds to a line in the image space [3]

Now problem turns into finding line intersections in parameter space. To solve it applies the following respectively. Once this problem is solved that means a line is detected.

- Divide the (m, n)-plane into a finite grid of cells
- Associate a counter c_{mn}, initialized by zero, with each cell - this yields the accumulation matrix $C = c_{mn}$
- For each image edge point p_i, we increment all counters on the corresponding line in parameter space, so counters at intersections will show higher counts than others.

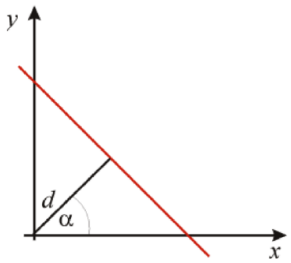


Figure 8: Representation in normal Form [3]

Lastly, this parameterization of the line in the image space has one problem and that is the line can be represented by infinitely many (m, n) so our parameter space is infinite. To overcome this, Hough transformation represents lines in normal form $d(\alpha) = x_0 \cos\alpha - y_0 \sin\alpha$

3.1.3 Optimizing and Displaying Lines

This part of the algorithm is more related to visualization. Without it, the code would give dashed, discontinued lines that depict the edges of the lines. Basically, dashed lines are taken and categorized as belonging to the left or right. Lines that have a positive slope belong to the right side of the lane or are assigned to the left side of the lane. Finally, by using the averages of right and left side averages, lane lines are calculated and drawn as shown in Figure 9.

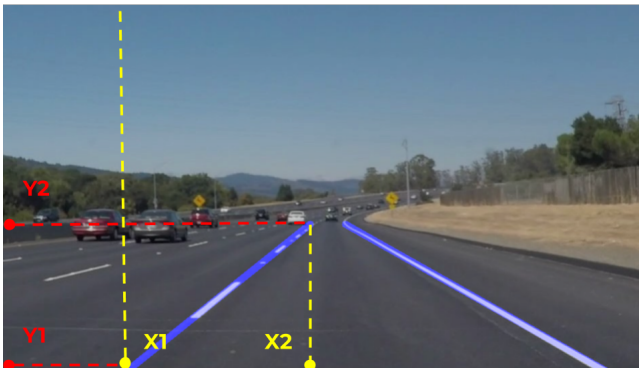


Figure 9: Calculation of Lane Lines [4]

3.2 Advanced Lane Detection Algorithm

This algorithm is more advance than the previous one, but again based on image processing techniques. The simple algorithm has poor performance in capturing the actual shape of the lane. Also could not get whole lane pixels, but lane lines' edge pixels. This is also not well enough to calculate lane curvature and vehicle position. So some advanced methods are needed to handle these problems. The structure of the advanced lane detection algorithm is given in Figure 10.

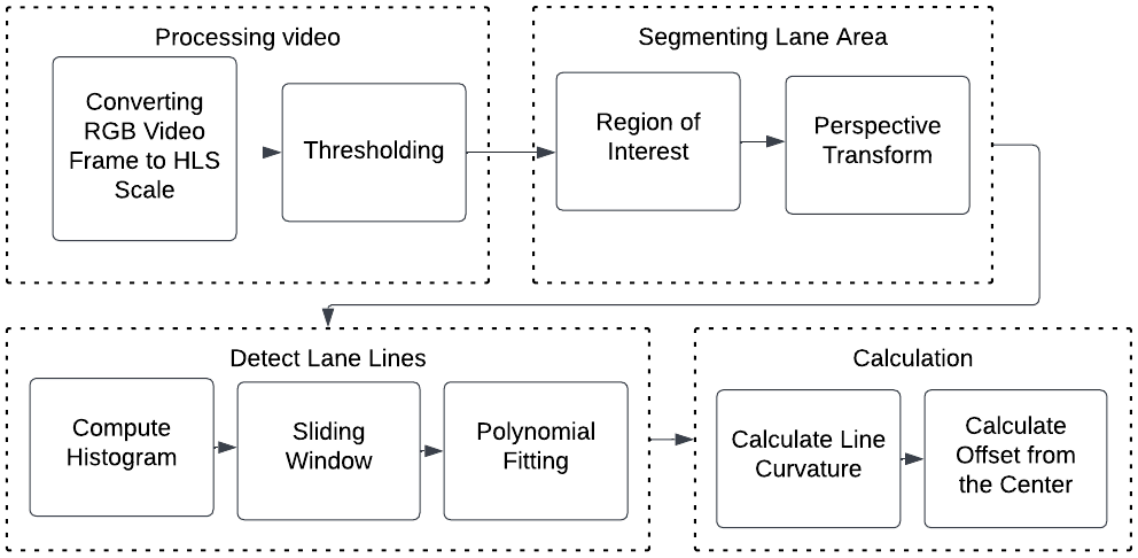


Figure 10: Advance Lane Detection Algorithm Structure

3.2.1 Thresholding

To detect lanes on the highway, the color and gradient thresholds are combined by following 3 stages as shown below:

- Apply Color Thresholding on Lightness Channel
- Sobel as a Gradient Threshold
- Combine Color Thresholding and Gradient Threshold

- **Color Thresholding** To detect objects of appropriate color values, it is needed to remove parts of the image outside the specified color range. To do that the color threshold is used. There are different types of color spaces such as Gray, HSV, HSL, HCL, HSB, and HSW. For RGB, sRGB(255,255,255) would produce a white image and recursively from the first component to the third component representing the Red, Green, and Blue colors. The aim is to find the right thresholds on the image to distinguish the yellow and white lines of the lane. It seems reasonable to use HLS(for hue, saturation, and lightness) to set thresholds for yellow and white lane lines.

- **Gradient Thresholding**

  Sobel operator is used to identify gradients, that is changes in color intensity in the image. Higher values represent denote strong gradients, and therefore sharp changes in color. [5]

  Sobel convolution kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels are applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these I_x and I_y) [5]. Then, these are combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by “(1a)” under section 2.1.2. The angle of orientation of the edge giving rise to the spatial gradient is given by “(1b)” [5]. The threshold is applied on gradient magnitude and gradient direction to get binary output images. Lastly, those binary images are combined together to get a combined gradient thresholding image.

- **Combined Thresholding**

  First, RGB thresholding was implemented on the red channel, and in the combined thresholding part (e.g see the third image in Figure 11)it seems a blue color.These give the noise on the sky part that is not being interested in and on the lane part gives a pink color that is combined blue and red color. For explained reasons, the lane information is only taken from HLS thresholding and Sobel thresholding. That means RGB color thresholding given blue color can be discarded to get rid of the noise. As a result, only a combination of the Color and Gradient thresholding gives the correct parameters to detect lanes in a robust manner.

Figure 11: Thresholding

3.2.2 Compute Histogram and Sliding Window

As you can see in Figure 12, the histogram of the Bird-eye view image have two peaks each representing one of the lane lines. The first peak represents the left line and the second one shows the right line. The histogram is only applied just on the bottom half of the image.

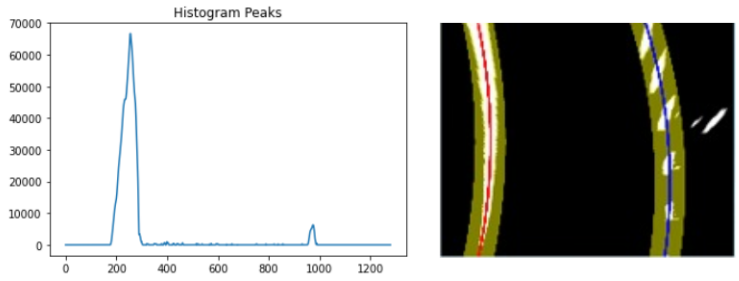


Figure 12: On the left histogram of the birds-eye view, on the right processed birds-eye view image based on histogram

The sliding window technique takes these peaks as a starting point for searching the line pixels. This technique basically looks for nonzero pixels inside the predefined window. Each window can be described via the number of windows that needs to be fit to the height of the image, and window width. Once it found pixels more than the predetermined threshold that belong to the lane pixels, it finds their average positions and takes this as a starting point for the next window. The search starts from the bottom and goes upwards. In the end, using these pixels, the algorithm fits a second-degree polynomial for each left and right lane line.

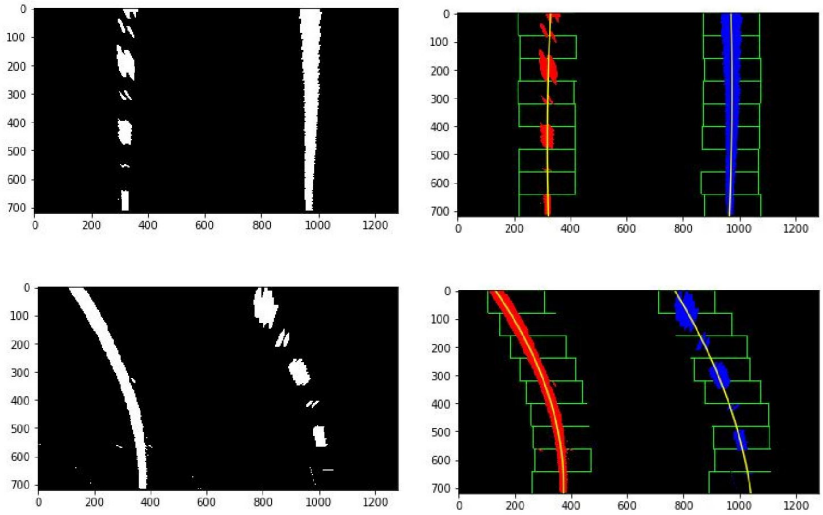


Figure 13: The improved sliding window algorithm [6]

3.2.3 Calculate Line Curvature and Offset from the Center

By using the location of the two detected lane lines found by sliding windows, calculate the curvature of the road and the car positions relative to the road can be calculated. First, it is assumed that the car camera is middle of the road. To find the curvature that could be a tangent to our lane lines, the y-value is defined, and find the max y-value to the bottom of the image. For implementation, the curvature calculation formula as given by:

$$\text{Radius of curvature } = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

On a straight lane the radius would be quite big, otherwise on a road has a high curvature, the radius would be small. Pixel space should be converted to meters by defining the appropriate pixel height to lane length and pixel width to lane width ratios. In our case, it was given height ratio: 32 meters / 720 px, width ratio: 3.7 meters / 800 px. [1]

The car's position and distance from the center are also computed. The Center point is calculated by finding the average coordinates for the left and right lines. Car distance from the lines is calculated by subtracting the middle point from lines as an offset and multiplying by the lane's pixel to real-world width ratio. [1]

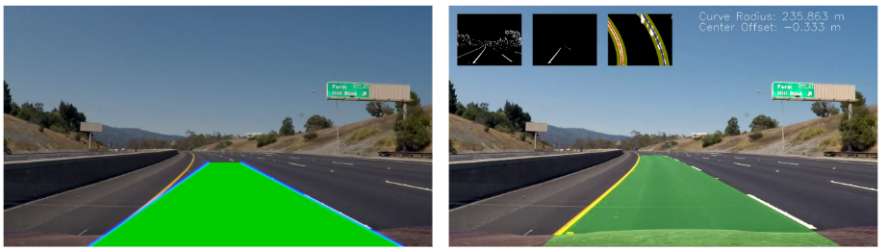
4 Results

In this section, the detection algorithms are tested in 3 different types of video in terms of process speed and the total number of correct detected frames. Two of the videos are in the highway conditions, lanes generally are pretty evident and visible. A distinction between them is one of them includes lane changes but most of the time road condition is the same. The other one does not include lane changes but road condition changes from time to time, and some shades of trees and objects are included in the road. Lastly, the third video is in mountainous conditions. It has harsher conditions for the implemented detection algorithm like occlusions, absence of lane line, sudden and big brightness changes, and high curvatures. Comparison results can be seen in Table 1.

Video		Simple Algorithm		Advance Algorithm	
Video	**Size**	**Correct/Total Fr.**	**Time**	**Correct/Total Fr.**	**Time**
Hway(Lane Changes)	1920x1080	940/1199	0.018s	983/1199	0.429s
Hway(Diff Cond.)	1280x720	420/1260	0.008s	1182/1260	0.192s
Mountain	1280x720	143/1199	0.011s	174/1199	0.196s

Table 1: Summary of experimental results.

The advanced algorithm is way better than the simple algorithm in terms of performance. Advance one can cover lanes way better because it uses second-order polynomials for each lane line, so can represent turns more decent. Also, we can find the curvature of the lane and car offset thanks to the advanced algorithm. Figure 14 shows the results of both algorithms results on the same place. In terms of working speed, a simple algorithm is pretty good. The advanced algorithm, on the other hand, as you can see in Table 1. It can only give us 2-3 Hz information, so it can not be used for real-time operations.



(a) (b)

Figure 14: On the left, the simple algorithm could not get the lane properly. On the right advanced algorithm gives better results

Simple and advanced algorithms are both basically failed on mountainous video because both algorithms are based on image processing and are highly physical, so lane lines need to be seen clearly. As you can see in Figure 15, sudden peak brightness changes make the algorithms completely blind. Tree shades or other vehicles can cause occlusions on the lane lines and in that case, part of that line can not be seen, so the polynomial fit is not able to draw the correct line. Lastly, fallen leaves cover the lines which is also another problem.

For better and more reliable performance, methods based on Deep Learning could be an option to go for. The current state-of-the-art approaches are also based on deep learning. Well-known state-of-the-art(s-o-a) approaches are CLRNet: Cross-Layer Refinement Network [7] for Lane Detection which is trained on CULane [8] and TuSimple [9] and LLAMAS [10] datasets. SCNN: Spatial CNN which is trained on CULane [8]. TUSimple and CULane are the datasets that include images that have very hard conditions as in Figure 16. In this study case, implementations proposed in this paper do not work well with conditions like images given dataset such as occlusion, brightness changes, various weather conditions, different day times, heavy traffic conditions, bad roads without certain lanes, etc. However, if the train is executed on the agent with these datasets, the lanes can be found with more accuracy in harsher conditions and with less latency way [8].

In comparison, CLRNet adopts F1-measure as an evaluation metric which is COCO [11] metric. They also introduced a new metric mF1 for better comparison, where F1@50, F1@55, $\cdots$, F1@95 are F1 metrics when IoU thresholds are 0.5, 0.55, $\cdots$, 0.95 respectively [7]. To learn details about precision, recall, IoU, etc. [12] can be looked at.

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

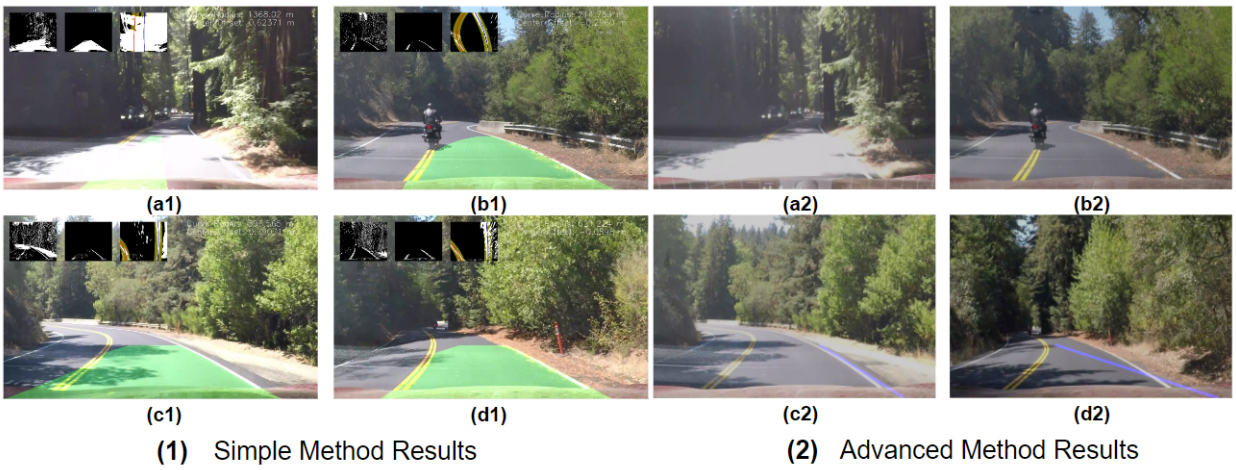


Figure 15: Harsh Conditions for both Algorithm

$$mF1 = (F1@50 + F1@55 + \cdots + F1@95)/10, \tag{3}$$

As Table 2 shown as, CLRNet perform both in terms of performance and also computation speed. Results are taken from the performance of both s-o-a approaches on the CULane dataset. SCNN can only give 7.5 Hz information which is not good for real-time application but usable. On the other hand, CLRNet gives approximately 100 FPS which is way better than for real-time applications. As a result, Deep Learning based approaches are superior in terms of performance and speed than the proposed algorithms in this paper which are based on image processing.

Method	**mF1**	**F1@50**	**F1@75**	**FPS**
SCNN	38.84	71.60	39.84	7.5
CLRNet	55.64	80.47	62.78	94

Table 2: Comparison of s-o-a approach performances on CULane dataset.

Figure 16: Difficult Situations [8]

5 Conclusion

This research shows the results of the 2 different image processing-based algorithms that are proposed. Also, two different current state-of-the-art approaches are compared as can be seen in Tables 1 and 2. It can clearly be seen that our approaches highly depend on the conditions of the road and lane lines should be visible all the time as opposed to the s-o-a approach. Because of that, conditions like rain, snow, sudden brightness change, occlusion, high curvature, and lane changing lower the accuracy of our algorithms. In terms of speed, our simple detection algorithm gives good results like the CLRNet. But, advanced detection algorithms could not keep up with them. So, it is basically not real-time capable. To improve the speed of the advanced algorithm, we can introduce lane detection by the previous frame, we can get rid of the histogram process at the beginning of each frame. To enhance the performance and speed of the advanced algorithm, using different techniques rather than the sliding window technique could be the better solution, and also third-order B-spline can be obtained to cover highly curvature roads [13]. To sum up, the implemented algorithms in this case study, especially the advanced algorithm, work well in sunny and highway conditions and when the lines are visible. But in case of needing more liability on any conditions, probably we need to move on with Deep learning-based approaches like CLRNet or SCNN.

References

[1] E. Forson, "Udacity self-driving car nanodegree project 1 — finding lane lines," available at https://medium.com/computer-car/udacity-self-driving-car-nanodegree-project-1-finding-lane-lines-9cd6a846c58c, 2017.

[2] ——, "Teaching cars to see — advanced lane detection using computer vision," available at https://towardsdatascience.com/teaching-cars-to-see-advanced-lane-detection-using-computer-vision-87a01de0424f, 2017.

[3] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[4] N. Ferdinand, "A deep dive into lane detection with hough transform," available at https://towardsdatascience.com/a-deep-dive-into-lane-detection-with-hough-transform-8f90fdd1322f, 2020.

[5] B. Hunt, "Computer vision: a first course: D boyle and r c thomas. published by blackwell scientific publications, uk. 1988. 210pp £12.95." *Image and Vision Computing*, vol. 8, p. 171, 1990.

[6] O. Gaskey, "Advanced-lane-detection," available at https://github.com/OanaGaskey/Advanced-Lane-Detection, 2019.

[7] T. Zheng, Y. Huang, Y. Liu, W. Tang, Z. Yang, D. Cai, and X. He, "Clrnet: Cross layer refinement network for lane detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 898–907.

[8] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[9] kivinju, "Tusimple competitions for cvpr2017," available at https://github.com/TuSimple/tusimple-benchmark/, 2017.

[10] K. Behrendt and R. Soussan, "Unsupervised labeled lane markers using maps," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[12] J. Hui, "map (mean average precision) for object detection," available at https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, 2018.

[13] J. Cao, C. Song, S. Song, F. Xiao, and S. Peng, "Lane detection algorithm for intelligent vehicles in complex road conditions and dynamic environments," *Sensors*, vol. 19, no. 14, p. 3166, 2019.